CSC411 Machine Learning
Project 3: Fake News

Ariel Kelman
Student No: 1000561368

Gideon Blinick
Student No: 999763000

19 March 2018

1 Dataset Description

Both datasets (real and fake) contain headlines about U.S. President Donald Trump. From the headlines, it is possible to determine with significant accuracy the probability that a given headline is fake or real based simply on the presence or absence of particular words in the headline. For example, after performing some preliminary analyses on the headlines, we discovered that the following 3 words might be of particular use.

1. The word "donald" appears 42.05% of the time in real headlines while appearing in only 17.55% of fake headlines. This 24.5% difference was by far the largest of any word.
2. The word "the" appears 27.9% of the time in fake headlines while appearing in only 7.9% of real headlines, for a difference of 20.02%.
3. The word "trumps" appears 11.1% of the time in real headlines while appearing in only 0.3% of fake headlines, for a difference of 10.8%.

The data was split into a training, validation, and testing set for analysis in the remainder of the report.

1.1 Results & Report Reproducibility

All results and plots can be generated using the code in `fake.py`. All code is python 3, run with Anaconda 3.6.3. Running the code will save all generated images in the `resources` folder, where they are used by LaTeX. Note that some of the sections require the code for other sections (in the same file) to be run first. To reproduce the report, simply run it through LaTeX. This will pull the most recently generated figures from the `resources` folder.

2 Naive Bayes Implementation

The Naive Bayes algorithm was implemented in three primary functions. In the first function `traing_part()` we use the training set to achieve estimates for p(real), p(fake), $p(word|real)$ and $p(word|fake)$. We estimated $p(word|real)$ and $p(word|fake)$ by counting the number of headlines in

the training set where the word appears and the headline is real (or fake), adding a factor of $m \cdot p$, and divided this result by the size of the training set plus m. We also included words that were not found in the training set (but found in validation or testing) and set their values to $\frac{m \cdot p}{count(headlines) + m}$, i.e. the value they would have from a count of zero in the training set. This function is `traing_part()`, which returns the values of p(real) and p(fake) and dictionaries with the remaining information.

In the next stage, we created a function `evaluate()` which uses the outputs of the first function - p(real), p(fake), $p(word|real)$, $p(word|fake)$ - to compute $p(headline|real)$ and $p(healine|fake)$ for a given set of headlines, and in turn uses those to compute $p(real|headline)$ and $p(fake|headline)$ for the set of headlines, which is the function output. To avoid "underflow" caused by the multiplication of many small numbers (the probabilities for each word are quite low, as most words appear in a small fraction of headlines), we utlize the fact that

$$a_1 \cdot a_2 ... a_i = e^{ln(a_1)+ln(a_2)...ln(a_i)}$$

in order to compute $p(headline|real)$ as the product of $p(word|real)$ for all words (whether or not the word appears in the headline).

The final step in the implementation the accuracy is checked (using the function `check_accuracy()` with a threshold of 0.5), and the parameters m and p were tuned to produce the best accuracy on the validation set. This was done via a simple search (see the function `optimize_mp()`). We began with a value of m $= 2 \cdot 5833$ and $\frac{1}{2 \cdot 5833}$ and optimized from there. This was chosen as our starting point as is 1 example per class per word, which is intuitively what adding in the m and p factors are trying to do - avoiding the issue of too few counts. Some of the values tested are shown in the following table.

mp	accuracy
$2 \cdot 5833$	83.1%
$3 \cdot 5833$	82.9%
$4 \cdot 5833$	82.7%

Table 1: Some tested values for optimizing Naive Bayes, and the corresponding accuracy on the validation set.

Our optimal m and p values were found to be 1 and 5833 (the number of words), respectively, achieving an acuracy on the validation set of 83.7%. These values gave an accuracy of 83.4% on the training set, while on the testing set we achieved an accuracy of 84.5%.

3 Predictive Factors for Naive Bayes

3.1 Words

The following 4 images present the top 10 words for each relevant probability. Note that for some classes, such as $p(real \| word)$, there were many other words that could have been in the list provided. Also note that for $p(real \| word)$ the probabilities are slightly greater than 1 due to rounding.

1. $p(real \| word)$

2. $p(real \| not\_word)$

3. $p(fake \| word)$

4. $p(fake \| not\_word)$

These values were obtained by applying Bayes rule. More specifically, we calculated $p(real \| word)$ by $\frac{p(word \| real) \cdot p(real)}{p(word)}$. $p(word \| real)$ can be obtained for every word by taking the number of headlines with a given word in the training set that are also real and dividing it by the total number of headlines that are real in the training set. $p(real)$ is the number of headlines in the training set that are real divided by the total number of headlines in the training set. Finally, p(word) is the number of headlines in the training set with the word divided by the number of headlines in the training set. All these are easily obtained.

To obtain $p(real \| not\_word)$, we use $\frac{p(not\_word \| real) \cdot p(real)}{p(not\_word)}$. p(real) is the same from the previous step, $p(not\_word \| real)$ is 1 - $p(real \| word)$, and $p(not\_word)$ is the number of headlines without a word in the training set divided by the total number of headlines. The calculcations for $p(real \| word)$ and $p(real \| not\_word)$ proceed in an equivalent way with an appropriate substitution of "fake" for any mention of "real".

What is observable from these results is that the precense of certain words has far greater predictive ability than the absence of any given word. This is logical, as certain words that appear rarely and only in either real or fake headlines will be entirely associated with the class they are found in. By contrast, the absence of any one word is not enough to definatively assign a class of real or fake since the remaining headlines without the given word do not all fall into one class or another.

4 Logistic Regression

Code from project 2 was modified to run simple logistic regression to predict whether a headline is real or fake. For each headline, a sparse vector was created, with 1's in each position that represents a word that's present in the given headline (the representation of this vector is discussed again in section 5). The outputs were passed through the softmax function to produce results that can be interpreted probabalistically.

A regularization term was added directly to the gradient function that penalizes the L1 norm of the θ's (the parameters of the regression). To determine the best values for the learning rate and the weighting of the L1 regularization term (parameterized by γ), a simple grid search was preformed (primarily focussed on varying γ), using the results on the validation set to determine the best hyperparameters. The learning curves for many of these tests can be found in the `resources/part4` directory.

The following learning curves shows the results for the hyperparameter settings $\gamma = 1$ with a learning rate of 10^{-3}, though the results did not vary much with changes to γ.

These parameters gave a final training accuracy of 98%, validation accuracy of 81.7%, and testing accuracy of 84.3% (likely due to the random division of data among the sets) after 1000 iterations.

5 Logistic Regression vs. Naive Bayes

When forming a prediction on a headline, both Naive Bayes and Logistic Regression effectively compute

$$\theta^T I = \theta_0 + \theta_1 I_1 + \theta_2 I_2 + ... + \theta_k I_k > thr$$

where thr is a given threshold value; in this project $thr = 0.5$. In both cases, the θ's represent the paremeters that define the prediction model, while the I's are a function of the input for which a prediction is produced.

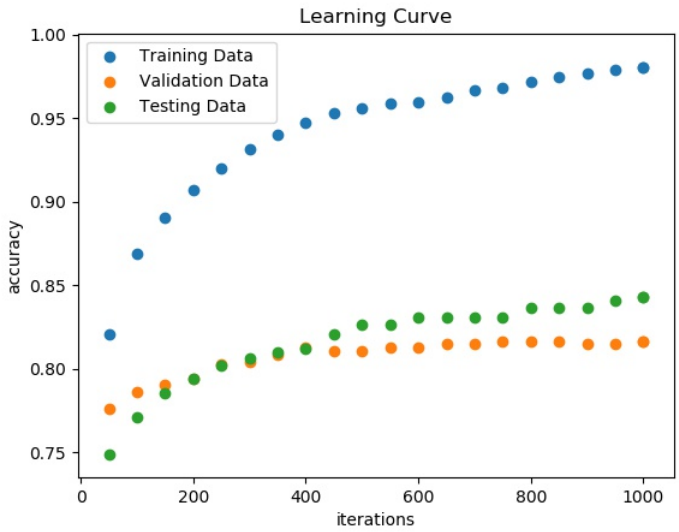

Figure 1: Learning curve for the logistic regression with a learning rate of 10^{-3} and $\gamma = 1$.

For logistic regression, each $I = I(x)$ for a headline is simply a vector of 0's and 1's, where each position in the vector represents a word that was included in the training set. A 1 indicates that the word is found in the headline, while a 0 indicates that one was not. As the headlines are relatively short in comparison to the total number of words, this vector is very sparse. The θ's represent how important a feature (the presence of a particular word) is to the final prediction (θ_0 is a bias that can account for the priors on real vs. fake headlines).

For naive bayes, consider the log-odds of a headline belonging to a particular class (real or fake). The classifier computes the log odds of each word appearing in a headline given each class ($ln\left[\frac{p(word|c)}{p(word|\ c)}\right]$; the bias is simply $ln\left[\frac{p(c)}{p(\ c)}\right]$). These log-odds are represented by θ, each θ_i is the log-odds of the corresponding word. As before, I_i is 1 when the word is in the headline and 0 otherwise.

6 Analysis of Logistic Regression

Table 2 shows the words that had the most impact on the classification of a headline as real or fake via logistic regression.

6.1 Ignoring Stop Words

If we eliminate the stop words in the above table, the following list shows the highest (in magnitude) values in *theta* and the corresponding words. (Note that the logistic regression itself still uses stop words, they are simply ignored in this analysis).

6.2 Analyzing Parameters

In the analysis of the logistic regression preformed above, the magnitude of the regression parameters were taken to signify the importance of the associated features (i.e. the presence of a particular word) to classification. In general this may be problematic, as different features may have different magnitudes (to take an example discussed in lecture, house prices and number of rooms may be features that are important to predicting selling price, but house prices are numerically much larger than the number

word	θ_i	word	θ_i
trumps	-5.2	breaking	2.63
us	-2.7	america	2.45
says	-1.87	hillary	2.37
comey	-1.79	just	2.28
turnbull	-1.76	u	2.14
ban	-1.71	black	2.06
donald	-1.68	victory	2.05
meeting	-1.68	are	1.99
korea	-1.68	watch	1.85
flynn	-1.67	an	1.81

Table 2: The words (and corresponding θ_i's) that have the most impact on classification via logistic regression.

of rooms in a house), in which case the magnitude of the parameters will differ from the effect they have on prediction. However, in the logistic regression for classifying headlines, all features have equal magnitude (when they are present). Therefore, no bias is introduced by these features that impacts on the classification other than the θ's - so the magnitude of θ_i is indicative of the importance of the corresponding word.

7 Decision Tree

7.1 Classification

Using the `sklearn` implementation of decision trees, we trained several decision trees to differentiate between the fake and real headlines. After some experimentation with the many parameters in the `sklearn DecisionTreeClassifier` (particularly with the maximum number of features used when looking for the best split), the default parameters gave the best results. As a split condition, maximum entropy was used rather than gini impurity, though similar results were obtained for both. Many of the settings served as ways to limit the size of the decision tree, so this result is not unexpected.

Decision trees with a maximum depth of $\{2, 3, 5, 10, 15, 20, 35, 50, 75, 100, None\}$ were built, and the results on the training, validation, and testing sets are shown in the figure below. The final point, not plotted, with no limit on the maximum depth, gave an accuracy of $1, 0.760.76$ on the training, validation, and testing sets respectively. As can be seen from the figure, the larger the depth of the decision tree, the greater accuracy achieved on the testing set. However, improvement is small on the validation and testing sets after a depth of 20. Without any other constraints (such as a minimum number of samples to split a node), a decision tree can reach perfect accuracy on the training set, as demonstrated above. Predictably, this leads to much greater preformance on the training set than on the validation and testing sets (though there is no *decrease* in preformance on the latter sets); showing the importance of using a validation and testing set to measure preformance.

word	θ_i	word	θ_i
trumps	-5.2	breaking	2.63
says	-1.87	america	2.45
comey	-1.79	hillary	2.37
turnbull	-1.76	just	2.28
ban	-1.71	u	2.14
donald	-1.68	black	2.06
meeting	-1.68	victory	2.05
korea	-1.68	watch	1.85
flynn	-1.67	putin	1.7
trump	-1.66	elect	1.67

Table 3: The words (and corresponding θ_i's) that have the most impact on classification via logistic regression, ignoring stop words.

7.2 Visualization

The following image shows the first few layers of the decision tree with depth 20. It was generated by saving a text representation of the tree (as a `.dot` file), which was then visualized using the `webgraphviz` tool (available at `http://webgraphviz.com/`).

It is interesting to look at the words being split on in these layers; X was a list of all words that appeared in either the fake or real datasets. These words are 'donald' (23), 'trumps' (1604; note the double appearance), 'the' (81), 'hillary' (44), and 'trump' (3). The exact same process of training was run ignoring stop words, but results were slightly worse; graphs similar to those mentioned above can be found in the `resources` directory (each filename indicates whether stop words were included, as well as info on the other non-default parameters). Note that because of the use of dictionaries (i.e. because `dict.keys()` does not keep track of order), running the code again may result in different indices for each of the above words.

A visualization of the entire tree is saves as `part7b_all.pdf` in the `resources` directory; the text represenations of many trees that were generated during training are saved in `resources/part7`.

7.3 Comparison of All 3 Classifiers

All three classifiers preform much better than random guessing, and quite well considering the difficulty of the task. Of the three classifiers, Naive Bayes had the best preformance (83.4%, 83.7%, and 84.7% on the training, validation, and testing sets respectively), slightly beating the logistic regression classifier (98%, 81.7%, and 84.3%). Though the difference is slight, it is notable that Naive Bayes achieves this accuracy despite making the extra assumption of conditional independence. It is also worth noting that Naive Bayes does not have significantly higher accuracy on the training set than on the validation or test sets. In last place is the decision tree mentioned above (depth 20, no limit on the maximum number of features used for splitting, and *entropy* as the metric for the quality of a split), which gave an accuracy of 83.8%, 73.7%, 75.7% on the respective sets.

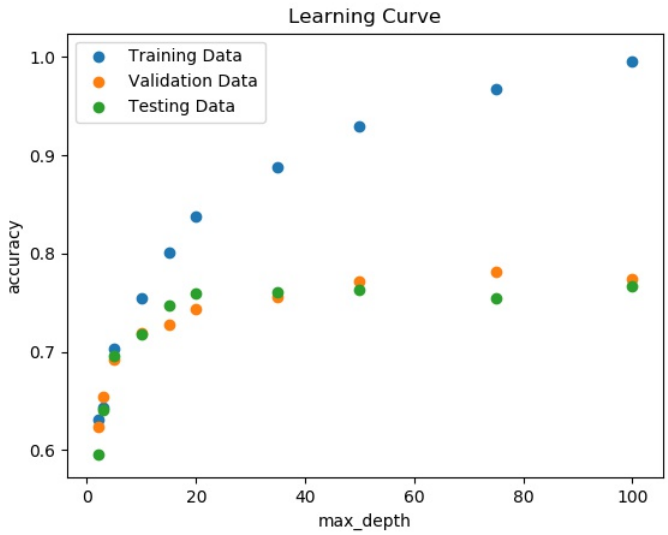

Figure 2: Plot showing the preformance of `sklearn` decision trees with varying depth. The split condition was maximum entropy, there was no limit on the maximum number of features for a split, and stop words were included.

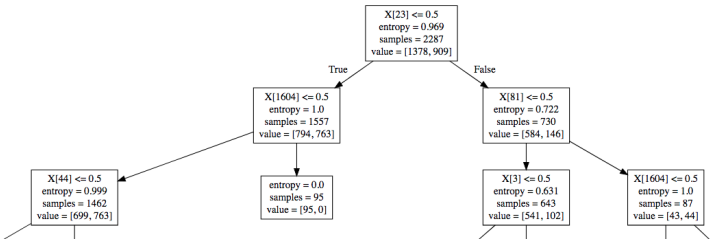

Figure 3: The top few layers of the decision tree (depth = 20).

8 Information Theory

8.1 Mutual Information on First Split

Using the result from part 7, the word that produced the best initial split was 'donald'. The mutual information (a measure of 'information gain') on that split (on the training data) can be calculated by

$$I(Y, x) = H(x) - H(x, Y) = H(Y) - H(Y, x)$$

where I is the mutual information, $H(x)$ is the entropy of x, and $H(x, Y)$ is the entropy of x conditional on Y. Plugging in the values shown in Figure 3 gives

$$\begin{aligned} I(Y, x) &= H(Y) - H(Y, x) \\ &= 0.969 - \Big[P_1(1) + P_2(0.722)\Big] \\ &= 0.969 - \left[\frac{1557}{2287}(1) + \frac{730}{2287}(0.722)\right] \\ &= 0.0557 \end{aligned}$$

where $x =$ '*donald*' and Y indicates whether a headline is real or fake.

A function `mutual_info()` was written to check this result, and to calculate the mutual information for other possible splits. The code for this function is shown below; it gave a result of 0.58 for $x =$ '*donald*'.

```
        def mutual_info(word, y):
            count_word = 0 #number of headlines with word in training set
            lines_with_word = []
            lines_without_word = []
            for k in range(len(training_set)):
                if word in training_set[k]:
                    count_word += 1
                    lines_with_word += [k]
                else:
                    lines_without_word += [k]
            prob_word = count_word/len(training_set)

            y_with = np.array( [y[i] for i in lines_with_word] )
            y_without = np.array( [y[i] for i in lines_without_word] )

            prob_fake = np.count_nonzero(y)/len(y)
            H = prob_fake*np.log(prob_fake) + (1 - prob_fake)*np.log(1 - prob_fake) #
    entropy before split
            H = - H/np.log(2) #convert to base 2, and apply negative

            prob_fake_with = np.count_nonzero(y_with)/len(y_with)
            Hy = prob_fake_with*np.log(prob_fake_with) + (1 - prob_fake_with)*np.log
    (1 - prob_fake_with)
            Hy = - Hy/np.log(2) #entropy of headlines with word

            prob_fake_without = np.count_nonzero(y_without)/len(y_without)
            Hn = prob_fake_without*np.log(prob_fake_without) + (1 - prob_fake_without
    )*np.log(1 - prob_fake_without)
            Hn = - Hn/np.log(2)  #entropy of headlines without word

            I = H - (Hy*len(y_with) + Hn*len(y_without) )/len(y)
            return I
```

8.2 Mutual Information on Later Split

The same procedure can be followed to compute the mutual information for a split on another word. Choosing a word randomly, gave a value of 0.0013 for the mutual information of a split of the training set on the word 'ways'. As expected, this value is lower than the mutual information for a split on the word 'donald' - that word was chosen as the first split precisely because it gave the most information about whether a headline was real or fake.